|  |
| --- |
| **МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**  Федеральное государственное бюджетное образовательное учреждение  высшего профессионального образования  «**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**  **ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»** |

|  |  |  |
| --- | --- | --- |
| Институт | – | Институт кибернетики |
| Направление | – | Информатика и вычислительная техника |
| Кафедра | – | Оптимизации систем управления |

Отчет по лабораторной работе №3 на тему:

«ПРИМЕНЕНИЕ ИСКУССТВЕННЫХ НЕЙРОННЫХ СЕТЕЙ ДЛЯ ОБРАБОТКИ ИНФОРМАЦИИ»

Вариант heart{3}

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Студент гр. 8ВМ23 |  |  |  |  |  | М.С. Бородин |
|  |  |  | (Подпись) |  | (Дата) |  | (И.О. Фамилия) |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  | Проверил |  |  |  |  |  | А.А. Белоусов |
|  |  |  | (Подпись) |  | (Дата) |  | (И.О.Фамилия) |

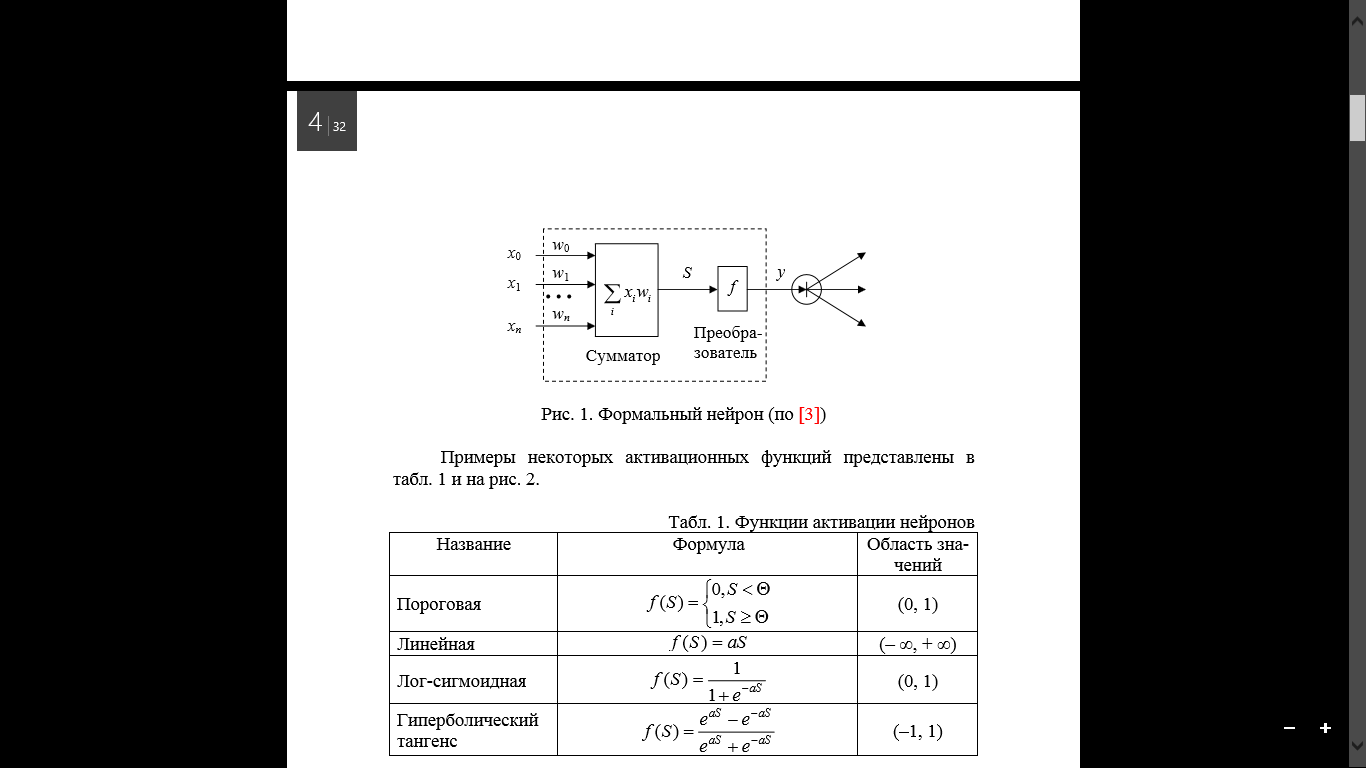
Томск – 2013

# Задание

С помощью нейронной сети с наличием как минимум одного скрытого слоя решить задачу заболевания сердца по входным данным.

# Решение

Нейронная сеть состоит из одинаковых нейронов. Каждый нейрон выглядит следующим образом:

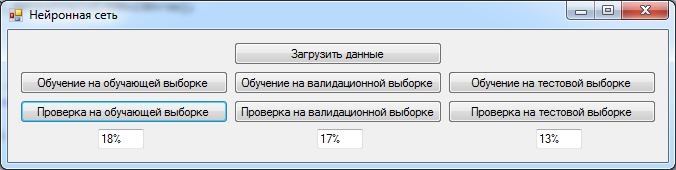


Нейрон сделан в виде класса Neuron. В нейроне содержатся веса связей, последние entrance значения, значение суммы и значение передаточной функции.

Нейронная сеть имеет 3 слоя: входной (35 нейронов), скрытый (18 нейронов) и выходной (2 нейрона).

# Реализация рещения

Вид формы:



# Код класса Network

public class Network

    {

        public Neuron[] InputLayer = new Neuron[35];

        public Neuron[] IntermediateLayer = new Neuron[18];

        public Neuron[] OutputLayer = new Neuron[2];

        public Network(int entrancesNumber)

        {

            var r = new Random();

            for (int i=0; i<InputLayer.Count(); i++)

                InputLayer[i] = new Neuron(entrancesNumber, r.Next(1000000));

            for (int i = 0; i < IntermediateLayer.Count(); i++)

                IntermediateLayer[i] = new Neuron(InputLayer.Count(), r.Next(1000000));

            for (int i = 0; i < OutputLayer.Count(); i++)

                OutputLayer[i] = new Neuron(IntermediateLayer.Count(), r.Next(1000000));

        }

        public double[] Entrance(double[] entranceValue)

        {

            foreach (var value in InputLayer)

                value.Entrance(entranceValue);

            var intermediateResult = new double[InputLayer.Count()];

            for (int i = 0; i < InputLayer.Count(); i++)

                intermediateResult[i] = InputLayer[i].ActivateFunction();

            foreach (var value in IntermediateLayer)

                value.Entrance(intermediateResult);

            intermediateResult = new double[IntermediateLayer.Count()];

            for (int i = 0; i < IntermediateLayer.Count(); i++)

                intermediateResult[i] = IntermediateLayer[i].ActivateFunction();

            foreach (var value in OutputLayer)

                value.Entrance(intermediateResult);

            intermediateResult = new double[OutputLayer.Count()];

            for (int i = 0; i < OutputLayer.Count(); i++)

                intermediateResult[i] = OutputLayer[i].ActivateFunction();

            return intermediateResult;

        }

        public void Teach(double[] entranceValues, double[] validValues)

        {

            var currentResultsOfNetworkWork = Entrance(entranceValues);

            var errorOutputLayer = new double[2];

            //ошибка для выходного слоя

            double alfa = 1;

            for (int i = 0; i < errorOutputLayer.Count(); i++)

                errorOutputLayer[i] = alfa \* currentResultsOfNetworkWork[i] \* (1 - currentResultsOfNetworkWork[i])

                    \* (currentResultsOfNetworkWork[i] - validValues[i]);

            var errorIntermediateLayer = new double[IntermediateLayer.Count()];

            //умножаем вес каждого нейрона на ошибку

            for (int i = 0; i < errorIntermediateLayer.Count(); i++)

            {

                double weight = 0;

                for (int j = 0; j < OutputLayer.Count(); j++)

                {

                    weight += OutputLayer[j].EntrancesWeight[i] \* errorOutputLayer[j];

                }

                errorIntermediateLayer[i] = alfa \* IntermediateLayer[i].ActivateFunction() \* (1 - IntermediateLayer[i].ActivateFunction()) \* weight;

            }

            var errorInputLayer = new double[InputLayer.Count()];

            for (int i = 0; i < errorInputLayer.Count(); i++)

            {

                double weight = 0;

                for (int j = 0; j < IntermediateLayer.Count(); j++)

                {

                    weight += IntermediateLayer[j].EntrancesWeight[i] \* errorIntermediateLayer[j];

                }

                errorInputLayer[i] = alfa \* InputLayer[i].ActivateFunction() \* (1 - InputLayer[i].ActivateFunction()) \* weight;

            }

            var h = 0.1;

            for (int j = 0; j < InputLayer.Count(); j++)

            {

                for (int i = 0; i < InputLayer[j].EntrancesWeight.Count(); i++)

                {

                    var delta = (-1) \* h \* errorInputLayer[j] \* InputLayer[j].EntranceValue[i];

                    InputLayer[j].EntrancesWeight[i] += delta;

                }

            }

            for (int j = 0; j < IntermediateLayer.Count(); j++)

            {

                for (int i = 0; i < IntermediateLayer[j].EntrancesWeight.Count(); i++)

                {

                    var delta = (-1) \* h \* errorIntermediateLayer[j] \* IntermediateLayer[j].EntranceValue[i];

                    IntermediateLayer[j].EntrancesWeight[i] += delta;

                }

            }

            for (int j = 0; j < OutputLayer.Count(); j++)

            {

                for (int i = 0; i < OutputLayer[j].EntrancesWeight.Count(); i++)

                {

                    var delta = (-1) \* h \* errorOutputLayer[j] \* OutputLayer[j].EntranceValue[i];

                    OutputLayer[j].EntrancesWeight[i] += delta;

                }

            }

        }

        public bool Validate(double[] entranceValues, double[] validValues)

        {

            var currentResultsOfNetworkWork = Entrance(entranceValues);

            var d = new double[2];

            if (currentResultsOfNetworkWork[0] >= currentResultsOfNetworkWork[1])

            {

                d[0] = 1;

                d[1] = 0;

            }

            else

            {

                d[0] = 0;

                d[1] = 1;

            }

            return ((validValues[0] == d[0]) && (validValues[1] == d[1]));

        }

    }

# Вывод

На практическом примере была реализована нейронная сеть для нахождения вероятности заболевания сердца Heart3. Задача была успешно решена, и после каждого обучения мы получали систему, работающую лучше, чем на прошлом шаге.